2차 보고서

Med-TENT: Source-Free Domain Adaptation을 활용한
전자 의료 기록(EHR) 텍스트 기반
질병 예측 모델의 병원 간 데이터 공유 한계 극복 및 성능 개선

Med-TENT: Overcoming Inter-Hospital Data Sharing Limitations and
Improving Text-Based Electronic Health Record Disease Prediction
via Source-Free Domain Adaptation

그로쓰 **17**팀 그린데빌

박선우(2171015)
유승혜(2190050)
전지윤(2270068)

1. Project-Summary (과제 요약)

1.1. 문제 정의

목표 사용자
본 연구의 목표 사용자는 병원 간 데이터 공유 및 활용에 제약이 있는 의료분야 종사자들(병원 관계자, 연구자 등)이다.

문제점

1. 데이터 이질성으로 인한 병원 간 **Domain Shift**
   병원 간 환자 분포, 측정 기기, 진단 기준 등은 서로 달라서 동일한 질병에 대해서도 데이터 특성이 상이한 domain shift가 발생한다. 이는 한 병원에서 구축된 모델을 다른 병원에서 적용하는 경우에도 나타난다.
2. 의료 데이터 공유 제약
   의료 데이터는 환자 개인정보 보호와 관련된 법적 규제(예: HIPAA[1], GDPR[2], PIPA[3] 등)가 엄격히 적용된다. 이러한 규제로 인해 개인의 건강 정보 및 의료 데이터의 직접적 공유가 어렵다.

이러한 요인들은 공유된 의료 AI 모델의 성능 저하로 이어지며, 의료 도메인의 인공지능 연구에서 극복해야 할 문제로 다가온다

필요기능

1. 병원 간 데이터 이질성을 극복할 수 있는 도메인 적응 기법이 필요하다.
2. 데이터 공유 없이도 기존 모델의 성능을 유지할 수 있는 방법이 필요하다.
3. 기존 부족했던 의료 텍스트 데이터 대상의 도메인 적응 연구 접근이 필요하다.

1.2. 기존 연구의 한계

기존 도메인 적응 연구는 주로 이미지를 대상으로 발전해왔으며, 의료 분야 역시 이미지 또는 생체 신호 데이터를 활용한 연구들이 주를 이룬다. ~~특히, Tent 가법은 이미지 데이터에 맞춰 개발된 모델이며, 해당 가법은 모델 불확실성을 측정하고 이를 관리하는 방법에 대해 명확한 방법론을 제시하고 있어 의료 이미지 데이터를 활용한 도메인 적응 연구는 많이 진행되었다.~~ 그러나 기존의 텍스트 데이터를 활용한 도메인 적응 연구에 비해 표준화된 연구 환경 및 데이터 접근성의 어려움이 존재하는 의료 텍스트 데이터를 대상으로 한 연구는 드문 편이다. 따라서, 의료 텍스트 데이터를 활용한 도메인 적응 연구는 아직 초기 단계에 있으며, 이와 관련된 연구를 진행하는 것이 중요한 도전 과제가 된다.

1.3. 제안 내용

연구 문제점 및 해결책
현재 데이터 공유 제약 문제와 도메인 적응 필요성에 대한 해결책으로, 데이터 공유 없이도 도메인 적응 기법을 활용하여 공유받은 기존 모델의 성능을 유지할 수 있는

---

[1] US, Health Insurance Portability and Accountability Act
[2] EU, General Data Protection Regulation
[3] KR, 한국 개인정보 보호법

방법을 제시하고자 한다. 구체적으로, 기존 모델을 다른 병원에서 사용하기 위해 데이터 공유 없이도 동일한 성능을 유지할 수 있도록 도메인 적응 기법을 적용하는 것이다.

기존 연구와의 차별성

1. 이미지가 아닌 텍스트 데이터 기반 도메인 적응
   기존의 도메인 적응 연구는 *Model Adaptation: Unsupervised Domain Adaptation without Source Data*[4]와 같은 연구처럼 이미지 데이터나 생체 신호 데이터를 중심으로 연구가 진행되었다. 그러나 텍스트 데이터는 이와 달리 표준화된 실험 환경 구축이 어렵고 모델 구조가 다르다. 본 연구는 이 차이를 좁히기 위해 텍스트 데이터에 적합한 새로운 도메인 적응 방법을 제안한다.
2. 의료 텍스트 데이터 대상 연구 한계 극복
   A Review of Recent Work in Transfer Learning and Domain Adaptation for Natural Language Processing of Electronic Health Records[5] 에서 볼 수 있듯 의료 분야에서 텍스트 데이터 대상의 연구는 부족함을 보인다. 기존 연구에서는 BioBERT, ClinicalBERT 등 의료 특화 모델 구축에 집중한 반면, 본 연구에서는 다루어지지 않았던 병원 간 domain shift 문제를 해결하고자 하였다. 또한 기존 연구에서는 반영하지 않은 실제 의료 환경의 제약-의료 데이터의 공유 제약과 라벨링의 시간적·인력적 비용 등-을 반영한 실험 설계로 현장 적용 가능성을 높이고자 하였다.

본 연구의 제안

이 문제를 해결하기 위해 본 연구는 텍스트 대상 도메인 적응 기법인 Med-TENT를 제안한다. Med-TENT는 질병 예측 모델이 공유되는 환경에서 소스 데이터 없이 다른 병원에서 예측 성능을 유지시키는 Source-free Domain Adaptation(SFDA) 알고리즘이다. 이는 기존 이미지 데이터 대상으로 개발된 Tent(Test-time Entropy Minimization[6], 이하 Tent) 기법의 아이디어를 차용하여 텍스트 데이터를 대상으로 적용하였다. Med-TENT는 LayerNorm parameter 학습을 바탕으로 Episodic mode와 dropout 비활성화를 통해 텍스트 데이터에 적합한 도메인 적응 모델로 동작한다.

1.4. 기대 효과 및 의의

기대효과

1. 도메인 차이 극복
   병원, 지역, 진료 환경 등에 따라 의료 텍스트 데이터의 표현 방식이 달라 발생하는 성능 저하 문제를 완화할 수 있을 것이다.
2. 데이터 공유 제약 해소
   원천 데이터를 직접 공유하지 않고도 테스트 시점에서 모델이 자체적으로 적응 가능하므로 데이터 공유 제약 문제를 우회할 수 있으며, 이를 통해 협력 연구나 다기관 연구에서 발생하는 실질적 제약을 줄이고 의료 AI 활용 가능성을 높일 수 있을 것이다.
3. 텍스트 기반 도메인 적응 연구 확장

---

[4] Rui Li, Qianfen Jiao, Wenming Cao, Hau-San Wong, Si Wu, https://arxiv.org/abs/2502.19316
[5] Egoitz Laparra , Aurelie Mascio , Sumithra Velupillai , Timothy Miller, https://www.thieme-connect.com/products/ejournals/html/10.1055/s-0041-1726522
[6] Wang, Dequan, et al., "Tent: Fully test-time adaptation by entropy minimization", arXiv preprint, arXiv:2006.10726 , 2020.

임상 기록, 전자의료기록, 환자 보고서 등 실제 활용 가치가 높은 텍스트 자원에서 도메인 적응 가능성을 검증할 수 있을 것이다.

연구 의의

1. 연구 공백 해소
   의료 텍스트 데이터를 활용한 도메인 적응 연구는 상대적으로 드물기 때문에 본 연구가 해당 영역의 공백을 채우는 역할을 할 것이다.
2. 실질적 임상 적용 가능성
   다양한 병원과 환경에서 작성된 의료 기록에 대해 일관된 성능을 발휘할 수 있는 모델 개발은 실제 임상 활용에 필수적이다. 따라서 기관별 데이터 편차를 줄임으로써 의료 ai의 신뢰성을 높이고 임상 의사결정 지원 도구로서의 실용성을 증대할 수 있을 것이다.
3. 도메인 적응 방법론의 일반화 가능성 제시
   본 연구에서 제시하는 방법은 의료 도메인에 국한되지 않고 법률 문서, 소셜 미디어 등 다양한 전문 텍스트 데이터 영역에도 확장 가능하다.

1.5. 주요 Task

1. Test-Time Entropy Minimization(Tent) 기반 적응 기법 적용
- 원래 이미지 데이터에 특화되어 제안된 Tent를 텍스트 기반 의료 데이터에 맞게 변형하여 적용한다.
- 사전 학습된 모델(Med-BERT)을 그대로 사용하고, 추가 학습 과정 없이 테스트 시점에서 바로 도메인 적응 가능하도록 수정한다.
- 엔트로피를 최소화하여 Source - Target 간 도메인 차이를 줄이고 더 신뢰도 높은 질병 예측 결과를 도출한다.
- 모델 전체를 학습하지 않고 일부 파라미터만 업데이트함으로써 효율성과 안정성을 확보한다.
- 기존 Tent는 BatchNorm 파라미터만 업데이트하지만,우리의 제안 모델인 Med-TENT는 텍스트 데이터 특성을 고려해 BatchNorm 대신 LayerNorm 파라미터를 조정하도록 수정한다.

2. 라벨에 의존하지 않는 적응 방식
- 실제 임상 환경에서 라벨이 없는 데이터가 대부분이라는 현실을 반영한다.
- 라벨 없이도 적응 가능한 Tent 구조를 활용하여 Med-TENT의 실제 적용성을 확인한다.

3. 실제 의료 데이터**(eICU)** 기반 검증
- 미국 중환자실 전자 의료 기록(EHR) 텍스트 데이터를 활용한다.
- Source 병원(7,059명 환자)과 Target 병원(723명 환자)을 도메인으로 설정하여 분포 차이가 있는 실제 상황을 모방한다.
- 텍스트로 기록된 환자 정보와 임상 기록을 활용하여 질병 예측 성능을 평가한다.

4. 비교 모델 설정 및 성능 검증
- 모델 1: 사전 학습된 Med-BERT 기반 질병 예측 baseline
- 모델 2(Med-TENT): 모델 1에 수정된 Tent 기법을 적용하여 도메인 적응 성능 보강

- 두 모델의 타겟 병원 데이터 예측 성능을 비교하여 우리가 제안하는 Med-TENT(모델 2)의 효과성을 검증한다.

5. 최종 목표
- Source 병원 데이터에서 학습된 baseline 모델 (Med-BERT) 대비 Target 병원 데이터에서의 Med-TENT(제안 모델) 예측 AUC와 AUPRC를 0.05 이상 향상을 목표로 한다.
- 기존 baseline 모델 대비 성능 개선을 입증함으로써 의료 텍스트 기반 도메인 적응 연구의 가능성과 임상적 활용성을 제시한다.

2. Project-Design (과제 설계)

2.1. 요구사항 정의

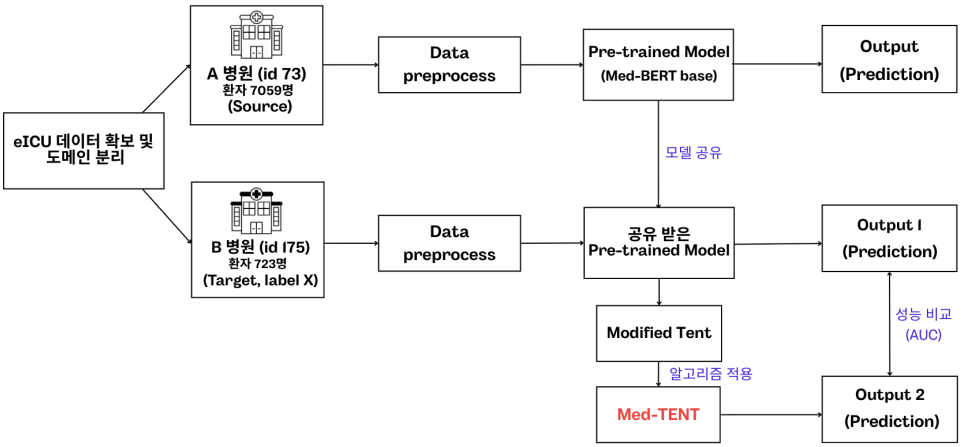
1. 사용 데이터
- 미국 eICU 데이터셋 활용: Source 병원과 Target 병원 선정한다.
- 병원 선정시 환자 수 및 지역적 분포 차이를 고려하여 도메인 간 이질성을 확보한다.
- 환자의 개인정보 보호를 준수한다.

2. 예측 모델 설계
- 사전 학습된 Med-BERT를 질병 예측 baseline 모델로 사용한다.
- 테스트 시점에서 추가 학습 데이터 없이 바로 도메인 적응 가능하도록 한다.
- 기존 Tent 구조를 텍스트 데이터에 맞게 수정: BatchNorm 대신 LayerNorm 파라미터를 업데이트하도록 코드 수정하여 우리의 모델인 Med-TENT를 제안한다.

3. 모델 예측 성능 평가
- Source 병원 데이터를 활용한 baseline 모델 성능을 확인한다.
- Target 병원 데이터에서 baseline 대비 Med-TENT 모델의 성능 개선을 검증한다.
- 도메인 차이로 인한 예측 성능 저하를 보완하는지 평가한다.

2.2. 실험 설계

[실험용 패혈증 예측 모델 실험 구성도**]**



1. 도메인 분리

사용 데이터셋인 eICU 데이터를 DB 내의 Hospital ID(병원 구별 용 ID)를 기준으로 하여 각 병원의 환자 수, 지리적 위치, 환자의 분포 등을 고려하여 2개의 실험 대상 도메인을 선정하였다. Source 도메인은 병원 ID 73의 병원(이하 73병원)으로 7059명의 환자 데이터를 포함하며, Target 도메인은 병원 ID 175의 병원(이하 175병원)으로 723명의 환자 데이터(Source 도메인의 10%)를 포함한다.

2. 데이터 전처리

각 데이터 별로 다음과 같은 전처리 작업을 수행하였다:

1) **Missing Data** 처리: 질병의 진단 코드인 ICD code가 입력되지 않은 경우 해당 데이터를 삭제한다.
2) 환자 별 입원 순서 추정: 환자 별 시계열 데이터를 처리해야 하기 위해 환자 별 입원 순서를 정렬한다. 단, eICU 데이터에는 입/퇴원 날짜 정보가 없어 순서가 추정 가능한 데이터들만 정렬하며, 불가능한 경우 임의의 한 입원만 포함시킨다.
3) 입/퇴원 날짜 입력: 환자 별 병원 방문의 입/퇴원 날짜를 입력한다. 단, eICU 데이터는 날짜 정보가 없어 기존 퇴원 년도와 총 입원 기간 정보를 통해 환자 별 첫 입원 날짜는 임의로 지정하였으며, 이후의 입원은 랜덤 간격을 두고 날짜를 지정하였다(입원 간의 간격은 모델의 고려 요소가 아니기에 무관하다). 상세 날짜 입력 로직은 아래와 같다.

- 최대 입원 기간 (totaloffset)
  - 병원 id 73: 9 months
  - 병원 id 175: 2 months

- Admission Year (hospitaladmityear)
  - 2014 or 2015
- Admission Date

  totaloffset(분) 기준으로 영역 별로 month 나눠서 임의로 지정
  - 병원 id 73
    - 9개월 ⇒ 1-3월
    - 5개월 (216,000) ⇒ 1-7월
    - 3개월 (120,000) ⇒ 1-9월
    - 1개월 이하 (약 40,000) ⇒ 1-12월
  - 병원 id 175
    - 1개월 (40,000) 초과 ⇒ 1-11월
    - 1개월 (40,000) 이하 ⇒ 1-12월
- Discharge Date
  - Year: hospitaldischargeyear (ehr에 있음)
  - Date: offset 계산

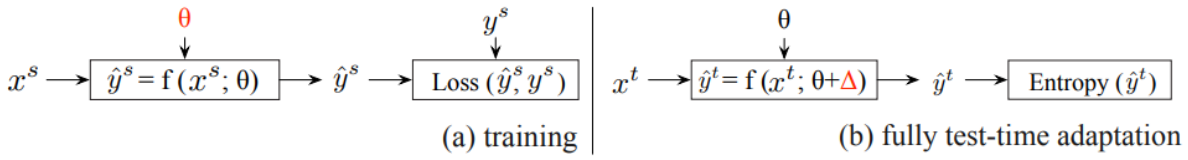
4) **ICD code** 통일: ICD-9 code와 ICD-10 code가 혼용된, 혹은 모두 사용된 eICU 데이터를 ICD-9 code로 통일시키기 위해 둘 다 사용된 경우 ICD-10 code는 삭제한다. 단, ICD-10 code만 사용된 진단의 경우, 직접 ICD-9 code로 전환하는 것은 적절하지 않다고 판단하여 해당 경우에만 ICD-10 code를 유지시킨다.
5) **Data Labeling**: 패혈증 여부에 대한 라벨링을 진행한다. 환자 별로 패혈증을 진단받은 경우 1, 진단 받은 경험이 없는 경우 0으로 라벨링한다.

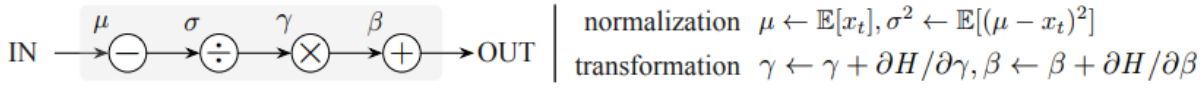
3. **Med-BERT Model Pre-training**
   73병원(Target)의 환자 데이터를 통해 Med-BERT 모델을 학습시킨다. 앞선 2. 데이터 전처리 과정을 바탕으로 BERT 모델의 입력 데이터 형식에 맞는 구조로 변환한 후, Pre-training을 통해 BERT 모델이 환자 별 진단 시퀀스를 학습한다.

4. **Med-BERT Model Fine-tuning (Baseline 구축)**
   Med-BERT로 생성한 임베딩을 입력받아 Logistic Regression 방법으로 73병원의 데이터를 바탕으로 환자의 패혈증 발생 여부를 학습한 후, Test 단계에서 이를 예측한다. 이 때, Test는 73병원의 Test dataset(Test1)와 175병원의 전체 데이터(Test2)를 대상으로 각각 시행하며, 이 때 175병원의 Test AUC와 AUPRC 값을 본 연구의 기준 성능으로 삼는다.
   핵심 **output: 73병원 Test AUC/AUPRC, 175병원 Test AUC/AUPRC(기준 성능)**

5. **Med-TENT 설계**
   도메인 적응 핵심 솔루션인 Tent 알고리즘을 텍스트 데이터에 맞게 수정한다. Tent는 엔트로피가 낮은 예측일수록 더 정확하지만, 소스-타겟 차이에 따른 데이터 손상은 엔트로피를 증가(예측 불확실성 증가)시킨다는 문제를 엔트로피 최소화(Test-Time Entropy Minimization)를 통해 보정한다는 아이디어의 도메인 적응 기법이다. 사전 학습된 모델을 사용하여 학습 과정 없이 테스트 시점에 바로 적용시킬 수 있으며,

라벨이 없는 상황에서 BatchNorm 파라미터만 업데이트하기에 일부 파라미터만으로 빠르고 안정적으로 적응 가능하다.





단, 기존 Tent는 이미지 데이터에 맞춰 고안된 알고리즘이기 때문에 eICU 텍스트 데이터에 맞게 조정할 필요가 있다. 따라서 본 연구에서는 기존 BatchNorm parameter를 대신하여 LayerNorm parameter를 업데이트하도록 수정하고, 텍스트 데이터에 적합한 방식으로 Tent 기법을 적용할 수 있는 Med-TENT를 설계하고자 한다.

6. **Adapted Model Test (최종 실험)**
   4에서 구축한 Baseline Model에 수정한 Tent를 적용한 Adapted Model인 Med-TENT를 구축하여 175병원의 전체 데이터(Test2)로 Test를 시행하여 AUC와 AUPRC를 구하고, 4의 baseline 기준 성능과 비교하여 성능 개선을 증명한다.
   기대 및 목표 성능: AUC/AUPRC 0.05 이상 향상

주요 연구 현황

1. 데이터셋 확보
   eICU 데이터셋 확보 및 초기 데이터 분석 실행하였다. 분석 결과 및 다기관 의료 데이터 특성을 반영하여 연구 실험 설계를 수립하였다.
   ~~output: 실험용 패혈증 예측 모델 시스템 구성도~~
2. 데이터 전처리 시행
   데이터 전처리 파이프라인을 구축하여 주요 변수 및 환자 단위로 데이터를 정리하였다. 해당 파이프라인을 기반으로 실제 전처리를 진행하여 학습 용 데이터 정제 완료하였다.
3. **Med-BERT** 기반 **Baseline Model** 구현
   Med-BERT Pre-training 및 Fine-tuning 진행을 통해 Baseline 모델 구현을 완료하였다. 해당 Baseline 모델을 활용한 실험을 진행하여 도메인 적응 알고리즘 적용 전 기준 성능을 확인한 결과, Source(175 병원)의 AUC가 Target(73 병원)의 AUC에 비해 약 0.4 저조한 것을 확인하였다. 또한, Source(175 병원)의 AUPRC도 Target(73 병원)의 AUPRC에 비해 0.3 정도 저하된 것을 확인할 수 있었다.
   Source(175 병원) Test_AUC1: 0.82, Test_AUPRC1: 0.46

		Test_AUC1		Test_AUPRC1	
		count	mean	count	mean
Model	Train_size				
Bert only	1878	10.0	0.820381	10.0	0.463684

   Target(73 병원) Test_AUC2: 0.49, Test_AUPRC2: 0.19

		Test_AUC2		Test_AUPRC2	
		count	mean	count	mean
Model	Train_size				
Bert only	1878	10.0	0.486278	10.0	0.188603

4. **SFDA** 알고리즘 적용 ~~(진행중)~~
   Tent 기반 SFDA 알고리즘 수정 및 적용한 제안 모델인 Med-TENT를 구축하여 baseline 대비 성능 비교를 진행하고 있다. 또한 Label smoothing 기법을 적용하여 불균형 데이터 환경에서의 모델 안정성을 개선하고자 시도하였다.

2.3. 실험 상세 구현

1. 데이터 전처리
   1) **Missing Data** 처리

```
# Remove rows with no ICD9 code
removed_hospital73 = hospital73.dropna(subset=['icd9code'])
```

Raw data인 'hospital73'에서 질병의 진단 코드 'icd9code'가 없는 행의 경우 해당 데이터를 삭제한다.

2) 환자 별 입원 순서 추정

```
# 2. 각 그룹에 대해 대표 퇴원 상태 판단
def is_expired_group(g):
    return ((g['unitdischargestatus'] == 'Expired') |
            (g['hospitaldischargestatus'] == 'Expired')).any()

group_metadata = df_dedup.groupby('group_id').first().reset_index()
group_metadata['is_expired'] = df_dedup.groupby('group_id').apply(is_expired_group).values

# 3. 환자 기준(uniquepid + 조건 일부)으로 퇴원 상태가 다른 그룹들을 모음
# 퇴원 상태만 다르고 나머지는 같은 그룹을 비교하기 위한 기준
base_group_cols = ['uniquepid', 'age', 'hospitaladmityear', 'hospitaldischargeyear', 'unitvisitnumber']
group_metadata['join_key'] = group_metadata[base_group_cols].astype(str).agg('|'.join, axis=1)

# 4. 각 join_key 그룹에서 expired 그룹은 유지, 그 외는 하나 랜덤 선택
def select_groups(g):
    expired = g[g['is_expired']]
    non_expired = g[~g['is_expired']]
    selected = []
    if not expired.empty:
        selected.append(expired)
    if not non_expired.empty:
        selected.append(non_expired.sample(1))
    return pd.concat(selected)

selected_groups = group_metadata.groupby('join_key', group_keys=False).apply(select_groups)

# 5. 최종적으로 선택된 group_id만 추출하여 원래 데이터에서 필터링
final_group_ids = selected_groups['group_id'].unique()
df_final = df_dedup[df_dedup['group_id'].isin(final_group_ids)].drop(columns='group_id')
```

한 환자 ID(uniquepid)에 대해 여러 개의 입원 기록이 존재할 수 있다. 즉, 하나의 uniquepid에 여러 patienthealthsystemstayid가 매칭될 수 있다. 그러나 입원 순서를 정확히 식별할 수 없는 경우가 있으며, 시계열 기반 모델 학습 특성상 이러한 상황에서는 해당 케이스들 중 임의로 하나의 입원 기록만을 선택해야 한다.

먼저 특정 기준(group_cols)에 따라 데이터를 그룹화한 후, 각 그룹 내에서 퇴원(Expired) 상태가 존재할 경우 해당 열을 우선적으로 보존하였다. 해당 그룹에 퇴원 기록이 없다면, 그룹 내 입원 기록 중 하나를 무작위로 선택하였다. 그리고 최종적으로 각 그룹별로 단 하나의 고유 ID(patienthealthsystemstayid)가 남도록 필터링하였다.

3) 입/퇴원 날짜 입력

```python
def make_date_row(row):
    year = int(row['hospitaladmityear'])
    disyear = int(row['hospitaldischargeyear'])
    totaloffset = row['totaloffset']
    month, day = random_month_day(year, totaloffset)
    adm_date = datetime(year, month, day)
    discharge_dt = adm_date + timedelta(minutes=int(totaloffset))
    discharge_dt = discharge_dt.replace(year=disyear)
    return pd.Series([adm_date.strftime("%Y-%m-%d"), discharge_dt.strftime("%Y-%m-%d")],
                     index=['admission_date', 'discharge_date'])

def add_dates_groupby_unit(df):
    # patientunitstayid 단위로 날짜 1세트 생성
    key_cols = ['patientunitstayid', 'hospitaladmityear', 'hospitaldischargeyear', 'totaloffset']
    uniq_units = df[key_cols].drop_duplicates()
    dates = uniq_units.apply(make_date_row, axis=1)
    uniq_units = pd.concat([uniq_units, dates], axis=1)
    df = df.merge(uniq_units[['patientunitstayid', 'admission_date', 'discharge_date']],
                  on='patientunitstayid', how='left')
    return df

hos73_final = add_dates_groupby_unit(hos73)
hos73_final
```

Raw data의 퇴원 년도 ‘hospitaldischargeyear’와 총 입원 기간 ‘totaloffset’을 바탕으로 입퇴원 날짜 정보(‘admission_date’, ‘discharge_date’)를 생성한다.

```python
from datetime import datetime, timedelta
def random_month_day(year, totaloffset):
    if totaloffset > 216_000:
        month = np.random.randint(1, 4)
    elif totaloffset > 120_000:
        month = np.random.randint(1, 8)
    elif totaloffset > 40_000:
        month = np.random.randint(1, 10)
    else:
        month = np.random.randint(1, 13)
    if month == 2:
        day = np.random.randint(1, 30) if year % 4 == 0 else np.random.randint(1, 29)
    elif month in [4, 6, 9, 11]:
        day = np.random.randint(1, 31)
    else:
        day = np.random.randint(1, 32)
    return month, day
```

환자 별 첫 입원 날짜는 임의로 지정하였으며(2.2.2의 로직 참고), 이후의 입원은 랜덤 간격을 두고 날짜를 지정하였다.

4) **ICD code** 통일

```python
def clean_icd_code(code):
    if pd.isnull(code):
        return code  # Leave NaNs unchanged

    # comma 기준으로 진단 코드 분리
    parts = [p.strip() for p in str(code).split(',') if p.strip() != '']

    # If any part is an ICD-9 code (usually all digits), keep the first such code
    for part in parts:
        if part.replace('.', '').isdigit():
            return part  # ICD-9
    # ICD-9 진단 코드가 없다면, 첫 번째 ICD-10 진단 코드 반환
    return parts[0] if parts else None

# icd9code 칼럼에 함수 적용
df['icd9code'] = df['icd9code'].apply(clean_icd_code)
```

Raw data의 '{ICD-9}, {ICD-10}' 형식으로 둘 다 사용된 경우, ICD-9만 보존한다. 단, ICD-10만 사용된 경우 이를 보존한다.

5) **Data Labeling**

```
# Label Sepsis
# 1 (case) : Patient with at least one Sepsis diagnosis
# (icd9=038.9)

# ICD-9 codes for Sepsis
disease_codes = ['038.9']

# get corresponding embedded codes
disease_tokens = []
for code in disease_codes:
  if code in types_dict:
    disease_tokens.append(types_dict[code])

print(f'disease tokens: {disease_tokens}')
```

```
disease tokens: [39]
```

패혈증 여부에 대한 라벨링을 진행한다. 먼저 대상 도메인 내에 존재하는 패혈증에 해당하는 ICD-9 code(038.9)가 임베딩된 토큰을 찾아낸다.

```
# labeling function
def code_processing(codes, visits):  # codes:
  found = 0  # flag whether the code is found
  new_codes, new_visits = [], []

  for code, visit in zip(codes, visits):
        if code in disease_tokens:
            found = 1
            continue   # pass the code/visit
        new_codes.append(code)
        new_visits.append(visit)

  return found, new_codes, new_visits
```

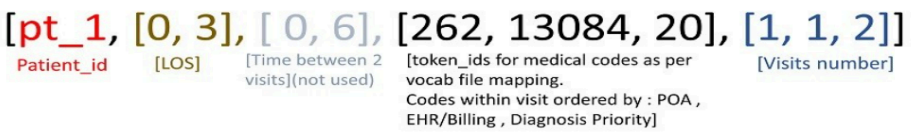
이후 해당 질병의 토큰(이 경우, 39)이 진단 시퀀스에 포함된 경우 1, 없는 경우 0으로 라벨링한다. 이 때, 기존 진단 시퀀스에서 해당 토큰 및 해당 진단 정보를 제거한다. 이는 추후 모델이 진단 시퀀스 사이의 관계가 아닌, 시퀀스에 토큰 포함 여부만으로 예측하도록 학습하는 것을 방지하기 위함이다.

2. Med-BERT Pre-training

본 연구의 사전학습 단계는 Med-BERT 공식 GitHub에서 제공하는 코드를 목적에 맞게 수정하여 수행하였다.

1) 소스 도메인으로 지정한 병원 ID 73의 데이터(Train, Test1)를 BERT 입력 형식에 맞추기 위해 Med-BERT에서 제공하는 preprocess_pretrain_data.py 스크립트를 활용하여 전처리를 진행하였다. 이 과정에서 환자별 진단 이력 시퀀스는 input_id, segment_ids, masked_lm_positions 포함하는 BERT 입력

구조에 맞춰 변환되었으며, 형태는 다음과 같다:



2) 이와 같이 준비된 입력 데이터와 함께 modeling.py, optimization.py, config.json, 그리고 run_EHRpretraining.py 코드를 활용하여 Med-BERT 모델의 사전학습을 진행하였다. 모델은 환자별 진단 시퀀스를 입력으로 받아, 각 시퀀스에 담긴 입·퇴원 시기와 진단 코드 등의 정보를 활용하여 진단 코드 간의 순차적 관계와 의미적 연관성을 학습한다.

```
***** Starting training loop *****
2025-10-21 02:12:37.085577: I tensorflow/core/kernels/data/tf_record_dataset_op.cc:381] TFRecordDataset `buffer_size`
Step 0 - Loss: 6.7643
Step 100 - Loss: 6.6736
Step 200 - Loss: 6.4415
Step 300 - Loss: 6.2706
Step 400 - Loss: 6.1369
Step 500 - Loss: 6.0321
Step 600 - Loss: 5.9491
Step 700 - Loss: 5.8839
Step 800 - Loss: 5.8306
Step 900 - Loss: 5.7861
2025-10-21 02:13:55.947695: I tensorflow/core/framework/local_rendezvous.cc:407] Local rendezvous is aborting with sta
***** Saving final model weights in both formats *****
Weights saved in Keras HDF5 format to: /content/pretrain_output/model.weights.h5
Weights saved in Checkpoint format to: /content/pretrain_output/model.ckpt-1
```

3) 의료 환경 특성상 데이터 분포가 불균형하다는 점을 고려하여 모델의 안정성을 향상시키고자 사전학습 단계의 run_EHRpretraining_utils.py 코드에 label smoothing 기법을 추가하였다. 이때 label smoothing 계수(ε)는 0.1로 설정하였다.[7]

```
# label smoothing 적용

epsilon = 0.1  # label smoothing 계수

one_hot_labels = tf.keras.layers.Lambda(
    lambda x: (1 - epsilon) * x + (epsilon / tf.cast(tf.shape(x)[-1], tf.float32))
)(one_hot_labels)
```

4) 이와 같이 사전학습된 모델의 성능 평가는 이후 테스트 단계에서 타겟 도메인으로 설정한 병원 ID 175의 데이터(Test2)로 이루어진다.

3. Med-BERT Fine-tuning

미세 조정 단계 또한 Med-BERT의 공식 GitHub에서 제공하는 predicting_DHF_MED_BERT_LR.ipynb 코드를 기반으로 하였다. 도메인 적응 실험을 위해 기존 학습 구조에 Tent(Test-time Entropy Minimization) 기법을 통합하기 위해 new_epochs_run() 함수를 새롭게 정의하였다.

1) new_epochs_run() 함수는 기존 baseline과 다르게 Tent 기법을 적용할 수 있도록 입력 데이터를 모델의 안정적인 적응을 위해 텐서 단위로 정규화하여

[7] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I., “Attention Is All You Need”, *Advances in Neural Information Processing Systems (NeurIPS)*, 30, –, 6000–6010, 2017.

처리했고, 각 테스트 배치에서의 적응 누적을 방지하기 위해 동일 가중치 모델을 반복적으로 로드하도록 설계되었다.

```
def new_epochs_run(make_model_fn, checkpoint_path, loader, lr, use_cuda=False, num_labels=1):
    """
    make_model_fn: 동일 구조 모델 생성 함수 (예: lambda: EHR_BERT_LR(...))
    checkpoint_path: fine-tuned best state_dict 경로
    loader: test2 DataLoader (my_collate 기반)
    num_labels: 1(시그모이드) 또는 2(소프트맥스)
    """
```

```
    # 1) 동일 가중치 모델 다시 로드 (적응 누적 방지)
    ehr = make_model_fn()
    state = torch.load(checkpoint_path, map_location=("cuda" if use_cuda else "cpu"))
    ehr.load_state_dict(state)
    if use_cuda:
        ehr.cuda()

    # 2) LN 기반 TENT 셋업 (HF 분류 모델에만 적용)
    base = ehr.PreBERTmodel
    disable_dropout(base)
    base = tentv2.configure_model(base)                # LN만 requires_grad=True
    ln_params, ln_names = tentv2.collect_params(base, top_k=4, include_classifier_bias=False)
    # optimizer = optim.Adam(ln_params, lr=lr)
    optimizer = optim.SGD(
      ln_params,
      lr=1e-3,
      momentum=0.9,
      nesterov=True,
      weight_decay=0.0
    )
```

2) Layer Normalization 기반으로 도메인 적응 기법을 수정하여 기존 Batch Normalization의 의존성을 제거하였으며, 테스트 시점에서도 Train 모드를 유지함으로써 적응과 추론 과정을 동시에 할 수 있도록 하였다.

```
# 3) 적응 + 추론 + AUC
y_true, y_hat = [], []
base.train()  # TENT는 test-time에도 train 모드 유지

for batch in loader:
    # my_collate → [all_ids, all_mask, all_segs, all_labels] 형태를 텐서로 정규화
    input_ids, attention_mask, token_type_ids, labels = unpack_from_mycollate(
        batch, use_cuda=use_cuda, num_labels=2
    )
    batch_dict = {"input_ids": input_ids, "attention_mask": attention_mask}

    if token_type_ids is not None:
        token_type_ids = torch.zeros_like(input_ids)

    outputs = tented_model(batch_dict)
    logits = outputs.logits.detach().cpu().numpy()

    if logits.ndim == 2 and logits.shape[1] == 1:
        # 시그모이드: B개
        probs = 1.0 / (1.0 + np.exp(-logits[:, 0]))
    else:
        # 소프트맥스: B개
        e = np.exp(logits - logits.max(axis=1, keepdims=True))
        probs = (e / e.sum(axis=1, keepdims=True))[:, 1]

    # 정답/예측 쌓기 (둘 다 반드시 길이 B)
    y_hat.extend(probs.tolist())
    y_true.extend(labels.detach().cpu().numpy().reshape(-1).tolist())
```

4. Med-TENT 방법

Med-BERT에 Tent(Test-time Entropy Minimization)를 적용할 때 나타나는 성능 불안정 문제를 완화하기 위해 우리의 모델인 Med-TENT를 설계하였다.

1) Med-BERT는 BatchNorm 파라미터가 아닌 LayerNorm 파라미터를 사용하고 있기 때문에 모델 적용을 위해 Med-TENT에서도 LayerNorm 파라미터만 테스트 시점에서 업데이트하도록 하여 도메인 이동에 따른 불안정을 줄이려고 노력했다.

```
def configure_model(model):
    """Configure model for use with tent (LayerNorm version)."""
    model.train()
    model.requires_grad_(False)   # 전체 freeze

    # LayerNorm만 학습 가능하게
    for m in model.modules():
        if isinstance(m, torch.nn.LayerNorm):
            m.requires_grad_(True)
            for p in m.parameters():
                p.requires_grad = True
    return model
```

2) ETTA[8] 또는 CoTTA[9]의 아이디어를 반영해 추가 Safe-TTA 게이트로 엔트로피 밴드를 설정하여 배치 평균 엔트로피가 너무 낮거나 높을 경우 업데이트를 스킵하고, 엔트로피 감소가 유의미하지 않을 경우 롤백하도록 하여 불필요한 적응으로 인한 성능 저하를 방지하였다. 또한, 엔트로피 최소화와 예측 다양성 최대화를 동시에 고려하는 정보 최대화(Information Maximization, IM) 손실을 결합하여 성능 붕괴를 방지하였다.

```
@torch.enable_grad()  # ensure grads in possible no grad context for testing
def forward_and_adapt(
    x, model, optimizer,
    T=1.3,           # temperature
    lam=0.5,         # diversity 가중치
    clip_norm=1.0,   # grad clipping (None이면 비활성)
    # --- Safe-TTA gates ---
    ent_band=(0.15, 0.60),  # 엔트로피 밴드 (binary일 때)
    div_floor=None,         # 다양성 하한 (None이면 비활성). multiclass는 자동 정규화.
    min_delta=1e-4,         # ΔEntropy 최소 개선폭(미만이면 롤백)
    eps=1e-8
):
```

---

[8] Niu, Shuaicheng; Wu, Jiaxiang; Zhang, Yifan; Chen, Yaofo; Zheng, Shijian; Zhao, Peilin; Tan, Mingkui., “Efficient Test-Time Model Adaptation without Forgetting”, *Proceedings of Machine Learning Research(ICML 2022)*, 16:16888–16905, 2022.

[9] Wang, Qin; Fink, Olga; Van Gool, Luc; Dai, Dengxin., “Continual Test-Time Domain Adaptation”, *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 7201–7211, 2022.

```
# --- 4) ΔEntropy 검사: 개선 미미하면 롤백 ---
with torch.no_grad():
    out2 = model(**x) if isinstance(x, dict) else model(x)
    logits2 = out2.logits
    if C == 1:
        p2 = torch.sigmoid(logits2 / T).squeeze(-1)
        ent_after = -(p2*torch.log(p2+eps) + (1-p2)*torch.log(1-p2+eps)).mean()
    else:
        probs2 = F.softmax(logits2 / T, dim=1)
        ent_after = -(probs2 * torch.log(probs2+eps)).sum(1).mean()

    if (ent_before - ent_after).item() < min_delta:
        # rollback
        for p, b in zip(params, backup):
            p.data.copy_(b)
        optimizer.load_state_dict(opt_state)
        return outputs  # 업데이트 부정효과 → 원래 출력 반환
```

3) Episodic 모드를 True로 설정하여 배치 간 누적 적응을 차단함으로써, 배치 순서에 따른 의존성을 최소화하였다.

```
def __init__(self, model, optimizer, steps=1, episodic=True):  #에피소딕 모드 변경
    super().__init__()
    self.model = model
    self.optimizer = optimizer
    self.steps = steps
    assert steps > 0, "tent requires >= 1 step(s) to forward and update"
    self.episodic = episodic
```

4) 테스트 데이터에 적용할 시에만 Dropout을 비활성화하여 모델의 일반화 성능에는 영향이 가지 않도록 안정적인 업데이트만 이루어지도록 하였다.

```
def disable_dropout(m):
  for name, child in m.named_children():
    if isinstance(child, torch.nn.Dropout):
        setattr(m, name, torch.nn.Identity())
    else:
        disable_dropout(child)
```

5) 계속되는 급격한 성능 저하를 줄이고자 Optimizer를 Adam에서 SGD로 변경하여 보다 보수적인 적응을 수행하도록 하였다.

```
optimizer = optim.SGD(
  ln_params,
  lr=1e-3,
  momentum=0.9,
  nesterov=True,
  weight_decay=0.0
)
```

6) 도메인 이동 시 과도한 업데이트로 인한 불안정을 억제하기 위해, 상위 L개(2-4개) 블록의 LayerNorm 계층 중 bias 파라미터만 업데이트하도록 제한하였다. 이를 통해 적응의 자유도를 최소화하고, 파라미터 분산을 줄여 모델의 안정성을 확보하였다.

```python
def collect_params(model, top_k=4, include_classifier_bias=False):

    for p in model.parameters():
      p.requires_grad_(False)

    L_total = model.config.num_hidden_layers
    start = max(0,L_total - top_k)

    params = []
    names = []

    for idx, block in enumerate(model.bert.encoder.layer):
        if idx < start:
            continue
        for subname, submod in block.named_modules():
            if isinstance(submod, nn.LayerNorm) and getattr(submod, "bias", None) is not None:
                submod.bias.requires_grad_(True)
                params.append(submod.bias)
                names.append(f"encoder.layer.{idx}.{subname}.bias")

    return params, names
```